# Testbeds Used For Exploring Learning from Observation

**Darrin C. Bentivegna**[1,2] and **Christopher G. Atkeson**[1,3]
[1]ATR-International, Information Sciences Division, Japan
[2]College of Computing, Georgia Institute of Technology, USA
[3]Robotics Institute, Carnegie Mellon University, USA
dbent@cc.gatech.edu, cga@cc.gatech.edu
www.cc.gatech.edu/people/home/dbent, www.cc.gatech.edu/fac/Chris.Atkeson

**Abstract**

This paper describes software and hardware environments that we use in exploring ways to have agents learn from observing humans. The two environments to be described are air hockey and the labyrinth marble maze game. While humans perform in the domain the behavior is observed and recorded. This data can then be used for many research endeavors such as seeding a numerical-learning algorithm or as input into discovering primitive/macro actions. For our research focus we are interested in using the captured data to teach an automated agent to operate in the environment.

## Introduction

Robots do not yet have the ability to observe a scene and automatically extract information needed for learning from the scene. For this reason, structured environments must be created that focus on collecting the data that is needed. This paper describes two environments that have been created to easily collect data while a human operates in a dynamic environment. In our research we are extracting primitives (Bentivegna & Atkeson 2000) from the captured data. The primitives are then used by automated agent to operate in the environment. One of the environments is playing air hockey. Figure 1 shows the physical implementation and figure 2 shows the simulation version. The other task used in this research is a marble-maze game that has also been implemented in a virtual environment and a physical implementation (figures 3 and 4).

These domains were chosen because of the ease with which they can be simulated in virtual environments and because they provide a starting point to obtain more information on learning from observation. The physical environments are also small enough to be operated in a laboratory. Since the basic movements in these domains are only in two dimensions, motion capture and object manipulation is simplified. A camera based motion capture system can easily be used to collect data in hardware implementations. A stationary arm or some other similar robotic device can be programmed to play air-hockey on an actual table.



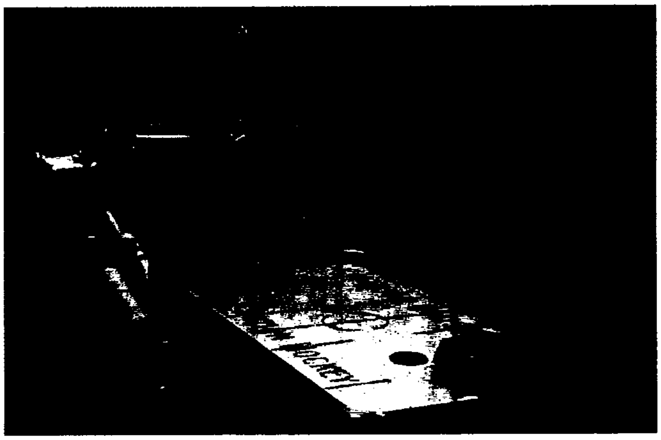

Figure 1: The hardware air hockey environment.

The software environments to be described share the same basic approach. The graphics are created using OpenInventor, the user interface uses Tcl/TK (Tcl/Tk 2000) and the low level control uses C and C++. The low level control can easily be used as functions to control objects in other graphic environments such as OpenGL. For these implementations the OpenInventor scene is rendered in an OpenGL window embedded in the Tcl/TK GUI using a program called TOGL (Togl 2000). Using this system allows the code to be run on any operating system that has OpenInventor installed on it. Windows and IRIX machines are currently being used, but it should also port seamlessly to Linux. If the OpenInventor code is replaced with only OpenGL calls, it could also be run on the Unix operating system.

## Air Hockey

Air-hockey, figure 1, is a game played by two people. They use round paddles to hit a flat round puck across a table. Air is forced up through many tiny holes in the table surface that create a cushion of air for the puck to slide on with relatively little friction. The table has an edge around it that prevents the puck from going off

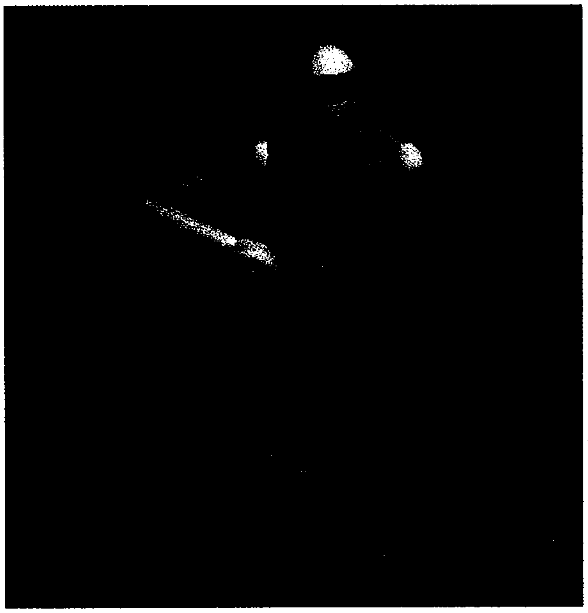

Figure 2: The virtual air hockey environment. The disc shaped object near the centerline is a puck that slides on the table and bounces off the sides, and the other two disc shaped objects are the paddles. The virtual player controls the far paddle, and a human player controls the closer paddle by moving the mouse.

of the table, and the puck bounces off of this edge with little loss of velocity. At each end of the table there is a slot that the puck can fit through. The objective of the game is to hit the puck so that it goes into the opponent's slot while also preventing it from going into your own slot. In our implementations we have eliminated the slot and designated an area on the back wall as the goal area. If it hits this area, a goal is scored. This simplifies the software simulation and reduces the need to chase the puck when a goal is scored in the physical environment.

## Hardware Air Hockey

The hardware air hockey implementation, figure 1, consists of the humanoid robot and a small air hockey table. The robot observes the position of the puck using its onboard cameras and hardware designed to supply the position of colored objects in the image. Because the humanoid's torso is moved during play to extend the reach of the robot the head is also moved so that the playing field is always within view.

**Computing Joint angles** This task uses 16 degrees of freedom. The following joints are used in the air hockey task:

- shoulder (2 joints),
- elbow,
- arm rotation,

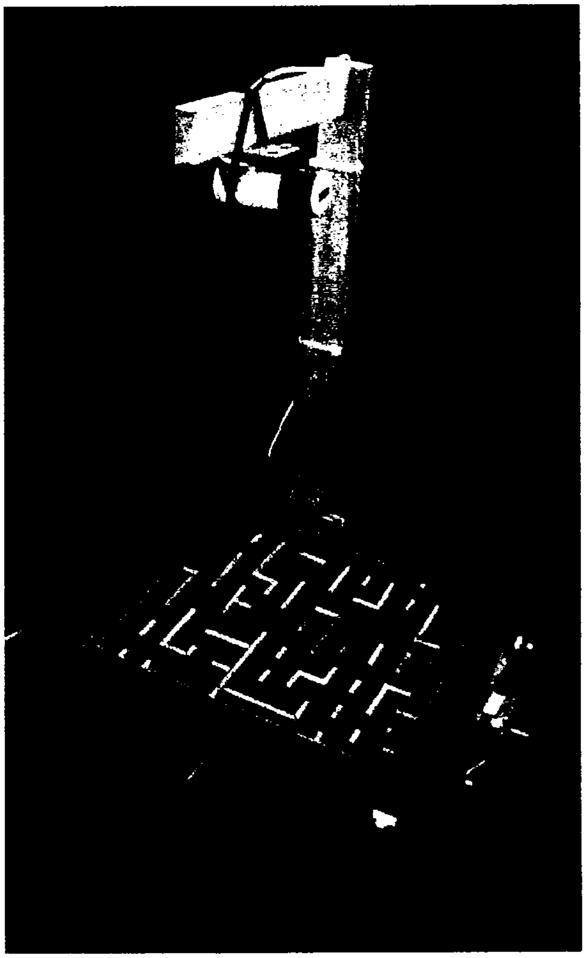

Figure 3: The physical marble maze implementation.

- wrist rotation,
- hand (2 joints),
- waist (3 joints),
- head (2 joints, lean and rotation),
- and eyes (2 joints each eye, pan and tilt).

Using all the joints above, except for the head and eyes, the robot must be positioned so that the puck is flat on the board and moves smoothly from one location to another. The other joints are used to position the head and eyes so that the entire board is in view at all times.

We have manually positioned the robot in several positions on the board while maintaining these constraints, figure 6. To get joint angles for any desired puck position, we interpolate using the four surrounding training positions and use an algorithm similar to that used in graphics for texture mapping (J. F. Blinn 1976), figure 5. This approach allows us to solve the inverse kinematics of the robot with extreme redundancy in a simple way. The four set positions surrounding the desired position define a polygon with Y up and X to the right. For desired position of P(x,y), as shown

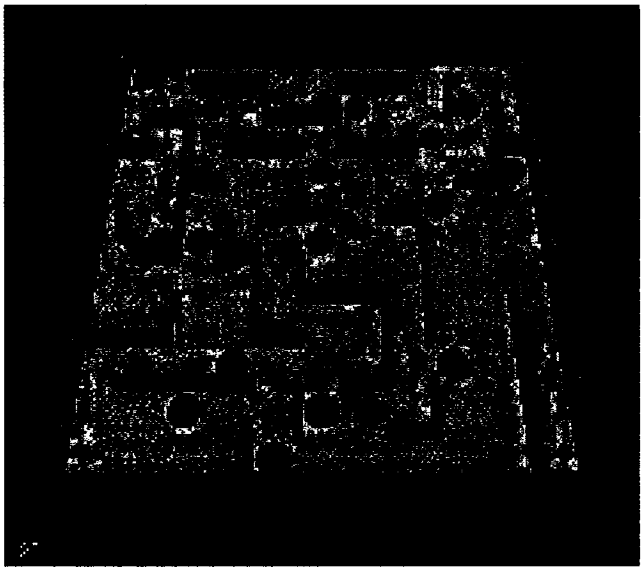

Figure 4: The virtual marble maze environment.

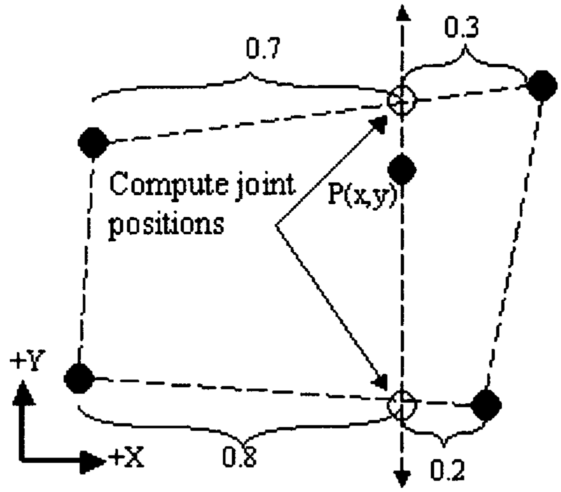


Figure 5: Using four given corner configurations to compute a configuration within the polygon.

in figure 5, a vertical line is drawn at P(x). Two joint configurations are computed at the top and bottom intersection of the line and the polygon. The joint angles are computed for the bottom-intersect location by using the two bottom joint configurations. These two configurations are averaged together and each one is weighted by the percentage from one point to the other. The configuration for the top point is computed in the same way using the top two corners of the polygon. The percentage of the distance that P(y) is from the bottom intersect point to the top intersect point is computed. This value is then used as a weight to average the two previous computed joint configurations in the same way that the top and bottom configurations were computed.

The robot is moved by specifying a desired board location and the robot configuration is computed for this position. The joints are then incremented from the current angle to the new desired angles. The robot is commanded to a position 420 times per second. The larger the increment, the faster the robot will move. Passing a desired paddle movement speed along with the desired board location specifies the movement of the robot. The distance from the current paddle location to the desired location along with the desired speed determine the number of steps that will be taken for this move. The total change in joint angle is then divided into that many steps for each joint using a fifth order equation at each step to provide the joint increment.

**Vision** It is the job of the robot's vision system to provide the location of the puck and the two paddles on the playing field. Due to the movements of the torso, the head of the robot, and therefore the eyes, move. A simple object tracking method that takes into account the constrained playing field is used to compensate for this movement and provide accurate object tracking. The vision system consists of cameras located in the head of the robot, figure 7, and a color tracking system for each camera output. The system tracks the four corners of the board, the puck, and a paddle. Figure 8 shows the four corners and the puck displayed in the vision system. Using the four known corners and the fact they are in a plane, the location of an object within that polygon and on the plane can be computed. The state of the puck consists of its position and velocity. The puck velocity is numerically computed using filtered positions. The vision system runs at 60 frames per second and as long as the four corners are in view, board positions can be accurately computed. If a corner is obstructed for a few frames, the previous known location is used to compute the puck's position, but with degraded accuracy.

## Software Air Hockey

Figure 2 shows the virtual air hockey game created that can be played on a computer. The game consists of two paddles, a puck and a board to play on. A human player using a mouse controls one paddle. At the other end is a simulated or virtual player. The software for this game can be obtained at www.cc.gatech.edu/projects/Learning_Research/. The movement of the virtual player has very limited physics incorporated into it. Both the human and agent paddle movements are constrained to operate with a velocity limit. Paddle accelerations are not monitored and therefore can be unrealistically large. The virtual player uses only its arm and hand to position the paddle. For a given desired paddle location, the arm and hand are placed to put the paddle in the appropriate location, and any redundancies are resolved so as to make the virtual player look "human-like". If the target position

Figure 6: The six given configurations of the robot used to compute all enclosed configurations.

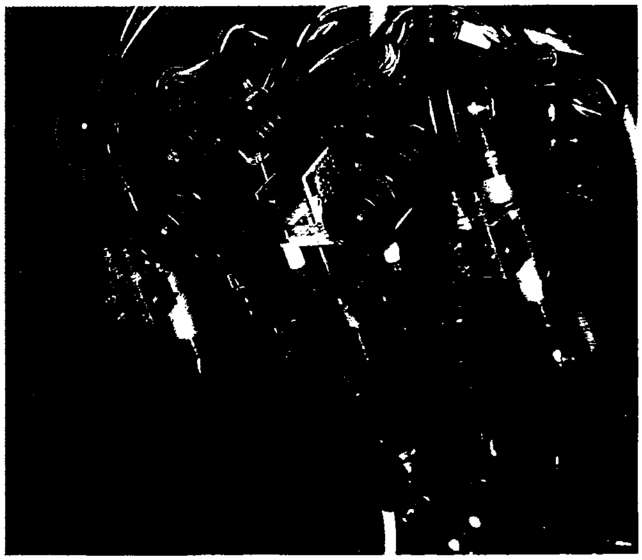

Figure 7: The head of the cyber human contains two "eyes," each made up of a wide angle and narrow angle camera on pan-tilt mechanisms.

for the virtual player's hand is not within the limits of the board and the reach of the virtual player the location is adjusted to the nearest reachable point on the board. The torso is currently fixed in space but could be programmed to move in a realistic manner. The virtual player's head moves so that it is always pointing in the direction of its hand, but is irrelevant to the task in this implementation.

The paddles and the puck are constrained to stay on the board. There is a small amount of friction between the puck and the board's surface. There is also energy loss in collisions between the puck and the walls of the board and the paddles. Spin of the puck is ignored in the simulation. The position of the two paddles and the puck, and any collisions occurring within sampling intervals are recorded.

New positions and velocities are computed at each frame. The time between frames, elapsed time, can be a set value, or it can depend on the speed of the computer. Using set times makes debugging easier and simplifies timing. The new velocity and location for the puck are computed using the following equations:

$PADVnew = PADV * (1.0 - friction * et)$

$PAD = PAD + 0.5 * (PADV + PADVnew) * et$

$PADV = PADVnew$

Since the walls are parallel to the coordinate axis, computing the effects of hitting the wall is simplified. When a wall collision has been detected, the velocity for the respective coordinate is changed as follows: $PUCV = -restitution * PUCV$, where restitution is a number less then 1.0.

Puck-paddle collisions are a little more difficult to compute. The angle between the puck and paddle's velocity vector must be computed at the time of impact. The collision vector is first rotated so that it is lined up with a coordinate axis, then the collision only affects one component and can easily be computed as follows:

$ball\_paddle\_distance = puckradius + paddleradius$

$cosine\_angle = (PUCx - PADx)/ball\_paddle\_dist$

$sine\_angle = (PUCy - PADy)/ball\_paddle\_dist$

$Vx = PUCVx - PADVx$

$Vy = PUCVy - PADVy$

$Vxrotated = cosine\_angle * Vx + sine\_angle * Vy$

$Vyrotated = -sine\_angle * Vx + cosine\_angle * Vy$

If the collision was against the puck (Vxrotated is less then zero) Vxrotated is adjusted as follows:

$Vxrotated = -restitution * Vxrotated$

The velocity must then be rotated back and added to the puck's velocity components.

$Vx = cosine\_angle * Vxrotated - sine\_angle * Vyrotated$

$Vy = sine\_angle * Vxrotated + cosine\_angle * Vyrotated$

$PUCVx = Vx + PUCKVx$

$PUCVy = Vy + PUCKVy$

## Marble Maze

In the marble maze game a player controls a marble through a maze by tilting the board that the marble is rolling on. The board is tilted using two knobs on the hardware version and using the mouse in the software version. There are obstacles, in the form of holes, that the marble may fall into. The ball travels in the X/Y direction on the Z plane and all the walls in the maze are parallel to the X and Y coordinates. The ball's movement in the X direction is controlled by a rotation about the Y-axis and movement in the Y direction is controlled by rotation about the X-axis. In

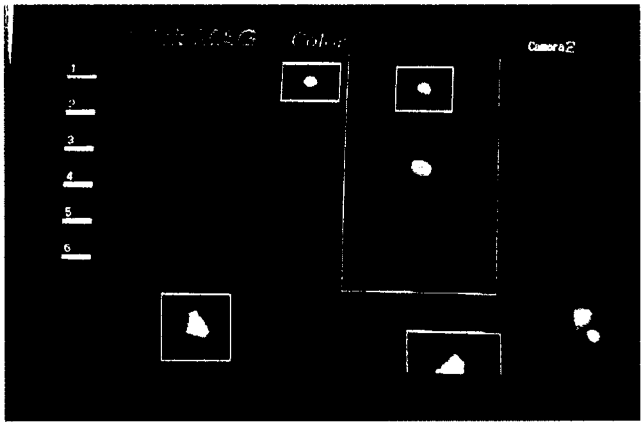


Figure 8: The four corners of the board and the puck in the middle as seen by the robot.

both versions the time and the board and ball positions are recorded as a human plays the game.

## Hardware Marble Maze

For the physical implementation of the marble-maze game, a Labyrinth game (Labyrinth 1999) has been outfitted with stepper motors to move the board and a vision system to supply the position of the ball. The movements of a human player can be captured and a computer can control the board. This setup was designed to be as small and simple as possible to allow the unit to easily be transported and controlled with a laptop running Windows. The system currently requires one serial port and one USB port.

**Motors and Sensors** The board is moved using two stepper motors, one for each axis. A board that communicates with the computer via a serial line controls the stepper motors. Boards from Pontech (Pontech 2000) and RMV (RMV 2000) have been tested and used. These boards provide the capability to read the position of the motor and send a position command to the motor within the time available.

There is a level sensor obtained from Spectron Glass & Electronics Inc. (Spectron 2000) connected to the bottom of the board that indicates when level position has been obtained. Due to the inaccuracies in the mechanical attachment of the sensor to the board, a calibration must be performed. The stepper motor controller boards also have digital input capabilities and therefore the level sensor is read via these boards.

Since the computer controls the board via the stepper motors, when a human plays there must be some way for them to indicate the desired board position. This is accomplished using two encoders that are connected to shafts with knobs on them. The encoders are read using a quadrature encoder reader from US Digital (US Digital 2000) that allows four encoders to be read via a serial port. As a human plays the encoders are read 60 times a second and converted to a motor command.

**Vision** As in hardware air hockey the vision system provides the location of the ball on the board. The system is simplified in this domain because the camera position does not change while the game is being played. Again to simplify the design, a Cognachrome vision system produced by Newton Research Labs (NewtonLabs 2000) is used. This system allows the image position of a colored object to be obtained via a serial port 60 times a second. To use this system the ball is colored so it can easily be seen. The system is first calibrated for camera distortion and hardware to software conversion. To calibrate for small changes in the camera position relative to the board, the system only needs to observe the ball in each of the four corners. The Cognachrome vision system provides the location of the marble within the image. This image coordinate is then converted to an undistorted image coordinate that is then converted to the actual board position. The ball's position is then filtered to smooth it out.

**Connection to Control Computer** As can be seen, there are three items that require RS232 serial communications. Since most computers are commonly configured with at most two ports, and laptops with only one, a USB to serial port adapter from Xircom (Xircom 2000) is used. This device connects to the USB port and provides four serial ports and two USB ports. Unfortunately the encoder reader will not function through this adapter and must be connected directly to the computer.

**Controlling the Board and Capturing Data** The code that controls the motors is contained in a C++ object that is ran as a high priority thread in Windows. The vision system outputs coordinates 60 times a second and therefore controls the timing. A function that reads the information from the Cognachrome via the serial port waits for the coordinates to arrive before continuing. While a human plays the game the coordinates of the ball are first read and converted to board coordinates. Then board and encoder positions are read. Finally the stepper motors are commanded to a position in accordance with the encoders. As long as the control functions are performed before the next vision coordinates are available, the system will run at 60HZ. The ball and board position along with the time are saved at each frame.

## Software Marble Maze

A software version of the marble maze game has been created by Silicon Graphics, Inc. and the software code is included as part of OpenInventor. This software is used as a basis for the simulator. The layout of the walls and holes are contained in a text file that can be read when the program starts. Start and end game position are also included in this definition file. The definition file allows the board to easily be reconfigured. The mouse controls the board position, which in turn controls the ball. The ball is simulated using a five-step process.

First the distance the ball moves in each direction is computed using the following equations.

$dX = et * Vx + et * GC * sin(-YrotationAngle)$
$dY = et * Vy + et * GC * sin(XrotationAngle)$

- et is the elapsed time between frames. It is set to a fixed time to reduce timing errors.
- Vx and Vy are the velocities of the ball in the X and Y directions respectively.
- GC is a number that represents the effect of gravity on the ball.
- rotationAngle is amount the board is tilted in each direction.

Next the new velocity and position of the ball is computed.

$Vx = dX/et$
$Vy = dY/et$
$tempX = X + dX$
$tempY = Y + dY$

The new ball position is checked to see if the ball has hit a wall. If a wall was hit, the ball is positioned against the wall and the new velocity is updated using the following equation for the appropriate velocity direction.

$V = V/ - WALL\_DAMPENING$

This has the effect of the ball bouncing off the wall with a loss of velocity of 1/WALL_DAMPENING. If the ball hits an edge, this equation must be computed for both directions.

Because the ball may have been repositioned back to its original location during the previous step, tempX is checked to see if it equals X. If so, Vx is set to zero. The same is performed for Vy and Y.

Finally the new position is checked to see if a hole exist in the area close enough for the ball to fall into. If it is, the ball continues to use the equations for motion in the XY direction and now the ball is also moved in the Z direction with velocity Vz that is initially zero according to the following equations with GC a constant the represents gravity:

$dZ = et * (Vz - et * GC)$
$Z = Z + dZ$
$Vz = dZ/et$

As can be seen from the simulation equations, the spin of the ball is not modeled and the ball is treated as if it were a sliding disk on a no friction surface. The time and the board and ball positions can be recorded at each frame. Other status such as the wall the ball may have hit or if the ball is falling into a hole is also saved.

## Results

The captured data from the software environments are currently being used in our research. The software environments allow us to easily collect data that is used to seed learning algorithms. We are just beginning to use the captured data from the hardware environments. Initially it appears to be of appropriate resolution and accuracy.

## Future Improvements

Some future modifications to the marble maze include attaching an inclinometer to the bottom of the board to increase the accuracy that the board position can be measured. Another change involves replacing the string drive system with chains and gears to reduce slippage during board movement. To reduce the reliance on the computer's serial port, the encoder reader will be replaced with a USB version or hardware will be created that will allow the encoders to be read via the stepper motor controller.

The board layout in the marble maze simulation is currently being modifying to allow the walls to be placed more accurately on the board. This will give us the ability to have the simulation environment more closely mimic hardware games. This requires the grid size to be much smaller so that walls may be accurately positioned within the environment.

## Conclusion

Capturing data in dynamic environments is very important for a variety of reasons. The construction of two dynamic environments in software and hardware has been shown. In these implementations data can be collected as the human performs in the environment. Our focus is to use this captured data to explore the use of primitives to teach hardware and software agents to perform the task. Our motivation is to reduce learning time needed by autonomous agents, create agents that have human-like behaviors, and obtain a better understanding of human learning mechanisms.

## Acknowledgements

This work is a joint project between the ATR-I Information Sciences Division, Georgia Institute of Technology and Carnegie Mellon University. It was also funded by National Science Foundation Award 9711770. The demonstration maze program that is included with OpenInventor was used as a basis for the virtual marble maze program.

## References

Bentivegna, D. C., and Atkeson, C. G. 2000. Using primitives in learning from observation. In *IEEE Humanoids2000*.

J. F. Blinn, M. E. N. 1976. Texture and reflection in computer generated images. *Communications of the ACM* 19(10):542–547.

Labyrinth. 1999. Labyrinth game obtained from the Pavilion company.

NewtonLabs. 2000. Newton Research Labs, Inc. homepage. http://www.newtonlabs.com.

Pontech. 2000. Pontech homepage. http://www.pontech.com.

RMV. 2000. RMV Electronics Inc. homepage. http://www.rmv.com.

Spectron. 2000. Spectron Glass & Electronics Inc. homepage. http://www.spectronsensors.com.

Tcl/Tk. 2000. Tcl/Tk homepage. http://www.scriptics.com.

Togl. 2000. Togl homepage. http://togl.sourceforge.net/.

US Digital. 2000. US Digital Corporation homepage. http://www.usdigital.com.

Xircom. 2000. Xircom, Inc. homepage. http://www.xircom.com.